Dealing with Multiple Failures in Zanshin: A Control-Theoretic Approach

Konstantinos Angelopoulos
University of Trento, Italy
angelopoulos@disi.unitn.it

Vitor E. Silva Souza
Federal University of Espirito Santo, Brazil
vitorsouza@inf.ufes.br

John Mylopoulos
University of Trento, Italy
jm@disi.unitn.it

ABSTRACT

Adaptive software systems monitor the environment to ensure that their requirements are being fulfilled. When this is not the case, their adaptation mechanism proposes an adaptation (a change to the behaviour/configuration) that can lead to restored satisfaction of system requirements. Unfortunately, such adaptation mechanisms don’t work very well in cases where there are multiple failures (divergence of system behaviour relative to several requirements). This paper proposes an adaptation mechanism that can handle multiple failures. The proposal consists of extending the Qualia adaptation mechanism of Zanshin enriched with features adopted from Control Theory. The proposed framework supports the definition of requirements for the adaptation process prescribing how to deal at runtime with problems such as conflicting requirements and synchronization, enhancing the precision and effectiveness of the adaptation mechanism. The proposed mechanism, named Qualia+ is illustrated and evaluated with an example using the meeting scheduling exemplar.

1. INTRODUCTION

Adaptive software systems are designed so that they fulfill their requirements in multiple ways (through multiple configurations/behaviours). When a failure is detected (i.e., a requirement of the system is not fulfilled), the system adapts by switching to an alternative behaviour. Unfortunately, such mechanisms treat each failure on its own merits and don’t work very well in cases where multiple failures occur.

The reason is simple: in case of failures F, F’, the chosen adaptations A, A’ may be conflicting, as A may call for a behaviour that exacerbates F’, and vice versa with A’. This problem is common in Control Theory, known as multivariable control problem for Multiple Inputs Multiple Outputs (MIMO) [5, 10, 17].

Current approaches either deal with requirement failures individually (e.g., Qualia [21]) or apply reconfiguration strategies using rules (e.g., Rainbow [6]). Both approaches have their pitfalls and drawbacks, as they can result in behaviour that continues to fail to fulfill requirements and/or is unstable because of existing dependencies among requirements. Therefore, while treating one failure by tuning some parameters of the system other requirements may be negatively impacted, resulting in immediate or eventual failure of other requirements rather than the one being treated. On the other hand, rule-based adaptation strategies, e.g., [13, 6], don’t scale very well as the number of combinations that need to be considered grows combinatorially with the number of concurrent failures. More specifically, if we had to define at design time adaptation rules for up to k possible failures then in order to cover all possible combinations of failures we would require O(2^k) rules.

The main objective of this paper is to propose an adaptation mechanism that can handle multiple failures (i.e., multiple failing requirements). Our proposal considers at the same time all failing requirements and attempts to select an adaptation that is coherent in the sense that it reduces the overall degree of failure, taking into account priorities among requirements. Our proposal supports the definition of adaptation requirements provided by stakeholders. For example an adaptation requirement may state that the adaptation should be conservative in that it does not change parameters in a way that could harm non-failing requirements. Such adaptation requirements are taken into account as the adaptation mechanism considers iteratively current failures, selects an adaptation, applies it, and observes results. The ultimate goal of this approach is to handle dependencies among requirements in a way that is consistent with stakeholder expectations about the adaptation process itself.

The rest of the paper is structured as follows. Section 2 presents the research baseline for this work. Section 3 describes the use of the Analytic Hierarchy Process (AHP) to prioritize requirements and the definition of adaptation requirements using concepts from the Zanshin framework. Section 4 depicts the Qualia+ adaptation mechanism and the extensions applied on Zanshin. In Section 5 we evaluate the new mechanism comparing it with older version of...
Qualia. Finally, in Section 6 we compare the related work with our proposal and concluding in Section 7.

2. BASELINE

This paper builds on concepts adopted from Goal-Oriented Requirements Engineering, such as goals for modelling stakeholder requirements, softgoals for modelling non-functional requirements, and AND/OR refinements that refine a goal $G$ into simpler goals $G_1, \ldots, G_n$ whose satisfaction implies the satisfaction of $G$. Tasks are actions that an actor can take (the system itself, or an external actor) to fulfill or operationalize a goal. For example, in Figure 1, Schedule meeting is a top-level goal, while Good participation is a softgoal. The goal Schedule meeting is AND refined into task Characterize meeting and subgoals Collect timetables etc. (4 subgoals in all).

Given this model, we use Zanshin [19] for the design of adaptive systems. Based on requirements and inspired by feedback control theory, the Zanshin framework consists of three steps: Awareness Requirements Engineering, System Identification and Evolution Requirements Engineering.

Awareness Requirements Engineering: the first step is concerned with the elicitation of Awareness Requirements, hereafter $AwReqs$, which are requirements that impose constraints over the success/failure of other requirements. $AwReqs$ determine the monitoring component of the feedback loop that monitors and adapts a base system. For instance, Figure 1 includes requirement $Q$: At least 90% of participants attend meetings. A possible $AwReq$ $AR6$ over this requirement is: $Q$ should have 75% success rate.

System Identification: the second step extracts indicators from elicited $AwReqs$ and identifies system parameters that, when changed, affect such indicators. These parameters can be either Control Variables (CVs) or Variation Points (VPs). The CVs are variables bounded within a limited range of values that could be either continuous or discrete. One the other hand, VPs are result of the OR-decompositions of the goal models and represent the functional variability of our system without though the extensive detail that architectural models do. The nature of this change is represented by differential relations. Back to the Meeting Scheduler example, indicator $I_6$ (meeting attendance) can be extracted from $AR6$. Then, parameter $FhM$, representing from how many participants one should collect timetables before scheduling a meeting, is identified. The impact that a parameter has on an indicator is characterized qualitatively by a monotonicity relation. For instance, the relation $\Delta (I_6/FhM) > 0$ represents the fact that if you increase $FhM$, $I_6$ should also increase (knowing more participants’ timetables results in better scheduling and, hence, better attendance). In this case the monotonicity of this relation is isotone. On the other hand, the relation $\Delta (I_6/FhM) < 0$ represents the fact that if you increase $FhM$, $I_6$ is expected to decrease (antitonic monotonicity).

Evolution Requirements Engineering: if $AwReqs$ are requirements for the monitoring component of the feedback loop, Evolution Requirements (hereafter $EvoReqs$) represent requirements for the adaptation component. $EvoReqs$ specify required changes to other requirements when certain conditions apply (e.g., the failure of an $AwReq$). For instance, say there is a domain assumption $D$ in the Meeting Scheduler’s requirements model that says Participants use the system calendar to keep track of their free/busy hours. If $AwReq AR7$: $D$ should always be true fails, one way of adapting is to replace assumption $D$ with a task $T$: Enforce participants’ use of system calendar. This requirement effectively defines an evolution of other requirements in the requirements model.

The Zanshin framework\footnote{See https://github.com/tefms-disi-unitn/Zanshin.} takes as input the requirements model for the base system, including the elements described above, and acts as a feedback controller that monitors failures and determines adaptation actions to be carried out. Zanshin supports two kinds of adaptation: evolution, described above, and reconfiguration. Reconfiguring consists on finding an alternative specification with new values for system parameters (identified during System Identification) in order to improve indicators of failing $AwReqs$ and take the system back to an acceptable state. For instance, if $AR6$ fails, which means $I_6$ needs improving, Zanshin might instruct the base system to increase $FhM$.

Although Zanshin can be integrated with other reconfiguration mechanisms, it provides its own named Qualia [21]. Qualia is an extensible adaptation mechanism that defines a basic algorithm composed of 8 procedures (parameter selection, parameter change, and so on), which can be replaced by more advanced procedures if desired. The basic algorithm uses qualitative information about indicators, parameters and their interrelations (defined by differential relations) in order to randomly select which parameter to tune from the set of parameters that can positively affect a given indicator of a failing $AwReq$. Advanced procedures may require more precise information from the model (e.g., parameters related to an indicator be given an order of magnitude of their effect), allowing Zanshin to handle multiple levels of precision for the adaptation mechanism.

This paper tackles one of Qualia’s main limitations: it deals with one failure at a time, on a first-come-first-served fashion. Moreover, it ignores the case in which the adaptation action chosen for one indicator affects negatively other indicators that its $AwReq$ may or may not be failing concurrently. For instance, for the Meeting Scheduler we identify the differential relations $\Delta (I_6/FhM) > 0$ and $\Delta (I_6/FhM) < 0$ that share the same parameter characterized by opposite monotonicity. Consequently if $AR3$ fails by decreasing the $FhM$ parameter, $I_6$ would be affected negatively.

3. REQUIREMENTS FOR ADAPTATION

Handling multiple requirements failures requires trade-offs. To achieve this, we extend Zanshin in a way that it can dynamically put together adaptation strategies using priorities over the requirements as criteria for resolving runtime conflicts among requirements that could not be eliminated at design time. We also propose the specification of Adaptation Requirements ($AdReqs$). These requirements are defined by reusing the Zanshin framework, namely the notation used to specify $AwReqs$ and $EvoReqs$.

3.1 Prioritizing Requirements

Requirements are prioritized in order to support the selection among alternative adaptations during the adaptation process. If $R$, $R'$ are both failing, it is important to know...
Characterize the system calendar participants (AR7).

NeverFail meeting participants (AR1).

Call participants NeverFail VPA Email comparisons is shown in Table 1.

For the purpose of illustration consider that we have four all the pairs of indicators and assign a value to each pair. Then, by using the scale presented in Table 2 we compare indicators includes the following steps. First, after the system identification process is carried out and the qualitative relations among indicators and the parameters are elicited, we identify which indicators present potential conflicts. By the term ‘conflict’ we mean that two indicators are influenced by the same parameter in opposite directions. For example if \( I_1 \) and \( I_2 \) are both influenced by parameter \( P_4 \) and \( \Delta (I_1/P_4) > 0 \) and \( \Delta (I_2/P_4) < 0 \) their differential relations mean that if \( AR1 \) and \( AR2 \) are failing and we can treat them only by tuning \( P_4 \) then we cannot fix both of them. Then, by using the scale presented in Table 2 we compare all the pairs of indicators and assign a value to each pair. For the purpose of illustration consider that we have four indicators \( I_1, I_2, I_3 \) and \( I_4 \) and the result of the pairwise comparisons is shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_1 )</td>
<td>1</td>
<td>3</td>
<td>1/5</td>
<td>1</td>
</tr>
<tr>
<td>( I_2 )</td>
<td>1/3</td>
<td>1</td>
<td>7</td>
<td>1/5</td>
</tr>
<tr>
<td>( I_3 )</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>1/5</td>
</tr>
<tr>
<td>( I_4 )</td>
<td>1</td>
<td>1/5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Pairwise Comparison Values

For an effective use of AHP we apply two heuristics that work as guidelines for assigning consistent priority values:

**Heuristic 1:** Indicators associated with hard-goals are preferred over those of soft-goals.

**Heuristic 2:** Indicators of hard-goals that are closer to the top-goal are preferred over lower level ones.

The purpose of these heuristics is to give higher priority to the functional integrity of the system over satisfaction of the non functional requirements. The process continues by calculating eigenvalues and then normalizing sums of rows. The final result is shown in equation 1.

\[
\frac{1}{4} \begin{bmatrix} 0.87 \\ 0.75 \\ 0.84 \\ 1.45 \end{bmatrix} = \begin{bmatrix} 0.22 \\ 0.18 \\ 0.21 \\ 0.36 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix}
\] (1)

We have now assigned weights over each indicator that represent their relative and we have a numerical guide to perform comparisons when needed. For instance, in the case where the system has to choose between fixing either \( I_1 \) and \( I_3 \) or \( I_4 \), even if \( I_4 \) is ranked higher than the other two their aggregated weight is higher and therefore should be preferred.

### 3.2 Adaptation Requirements

Our proposal includes a component that, given a requirements model and differential relations among requirements and system parameters is able to dynamically compose adaptation strategies that can handle multiple failures. As a first step we prioritized the indicators of the target system with weights that also measure their overall contribution to the correct operation of the system. This, combined with the qualitative relations from the system identification process allows Zanshin to automatically compose adaptation strategies, reconfiguring the system and maximizing the value of the satisfied indicators.
Our framework deals with the following kinds of failure:

- **Single Failure (SF)**: Only one indicator is failing and needs to be fixed.
- **Multiple Independent Failures (MIF)**: Many indicators are failing and either they don’t have any common parameters or the common parameters have the same monotonicity with every failing indicator.
- **Multiple Dependent Failures (MDF)**: Many indicators are failing and all or groups of them share parameters with opposite monotonicity.
- **Priority Conflict (PC)**: A failing indicator can be fixed only by tuning a parameter that harms a non-failing indicator of higher priority.
- **Synchronization Conflict (SC)**: An indicator is currently being treated and in the meantime a new failing indicator requires tuning of a parameter that has a negative impact on the one being treated.

In Control Theory a common way to resolve such conflicts is through compensation [4]. This method involves the addition of control mechanisms called compensators whose role is to compensate for deficient performance. In other words, compensators make trade-offs on output values or modify the reference input to achieve an acceptable result. Zanshin is able to accommodate compensation mechanisms by defining EvoReqs that refer to the adaptation process itself. In this work we exploit this capability of Zanshin to achieve conflict resolution and minimize the error caused by failing AwReqs.

Therefore, we model in Zanshin additional AwReqs and EvoReqs whose subject is the adaptation process itself, rather than the base-system. Figure 2 presents a simple goal model that prescribes how conflicts should be resolved during the adaptation process. The task Adapt Conservatively instructs the framework not to harm non-failing AwReqs while fixing the failing ones. The alternative task Adapt with Compensation represents the compensation mechanism we mentioned earlier. More specifically, the adaptation framework is allowed to harm a non-failing AwReqs of higher value for fixing another one, only if there is a possible action that would increase the indicator of the AwReq being harmed. Along the same lines, Adapt Optimistically does not consider priority conflicts as hazards for the base system because there is the assumption that the non-failing AwReqs are tolerant enough to a potential negative impact. We refer to these requirements as AdReqs.

<table>
<thead>
<tr>
<th>Relative Intensity</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Of equal importance</td>
<td>The two indicators are not conflicting</td>
</tr>
<tr>
<td>3</td>
<td>Slightly more important</td>
<td>Experience and judgement slightly favors one indicator over the other</td>
</tr>
<tr>
<td>5</td>
<td>Essentially more important</td>
<td>Experience strongly favors one indicator over another</td>
</tr>
<tr>
<td>7</td>
<td>Very much more important</td>
<td>An indicator is strongly favored and its dominance is demonstrated in practice</td>
</tr>
<tr>
<td>9</td>
<td>Extremely more important</td>
<td>The evidence favoring one over the other is of the highest possible validity</td>
</tr>
<tr>
<td>2, 4, 6, 8</td>
<td>Intermediate values</td>
<td>When compromise needed</td>
</tr>
</tbody>
</table>

Reciprocals: If indicator I has one of the above numbers assigned to it when compared with requirement I’, then I’ has the reciprocal value when compared with I.

### Table 2: Scale For Pairwise Comparisons

<table>
<thead>
<tr>
<th>Relative Intensity</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Of equal importance</td>
<td>The two indicators are not conflicting</td>
</tr>
<tr>
<td>3</td>
<td>Slightly more important</td>
<td>Experience and judgement slightly favors one indicator over the other</td>
</tr>
<tr>
<td>5</td>
<td>Essentially more important</td>
<td>Experience strongly favors one indicator over another</td>
</tr>
<tr>
<td>7</td>
<td>Very much more important</td>
<td>An indicator is strongly favored and its dominance is demonstrated in practice</td>
</tr>
<tr>
<td>9</td>
<td>Extremely more important</td>
<td>The evidence favoring one over the other is of the highest possible validity</td>
</tr>
<tr>
<td>2, 4, 6, 8</td>
<td>Intermediate values</td>
<td>When compromise needed</td>
</tr>
</tbody>
</table>

To monitor the success/failure of requirements for the adaptation process depicted in Figure 2 we use AwReqs. In this case AR2 imposes the constraint that non-failing AwReqs should not fail when receiving negative impact. As we would do for any goal model of a target system we define an adaptation strategy to overcome failures of this Awareness Requirement. Given the differential relation $\Delta (I_2/V P_1)$ [AdaptConservatively $\rightarrow$ Adaptwith Compensation $\rightarrow$ AdaptOptimistically] $< 0$ (the arrows indicate growing enumeration values [20]) the strategy switches among the possible values of the parameter VP1.

**Figure 2: Adaptation Requirements Goal Model**

**Listing 1: Adaptation Strategy for Adaptation Requirements**

Independently of the value of parameter VP1, the adaptation framework is required to perform trade-offs among indicators of failing and non-failing AwReqs. For some indicators there is a parameter to change in order to bring it closer to fulfillment. For others there won’t be any, at least ones that don’t harm higher priority indicators. Hence, when potential conflicts are detected during system identification process EvoReqs operations (e.g. abort, retry, replace etc) should be assigned to every indicator that may conflict with others. For instance, for the Meeting Scheduler AR6 and AR10 are both dependent on the parameter...
FhM through the differential relations $\Delta (I_0/FhM) > 0$
and $\Delta (I_10/FhM) < 0$. Consequently, if $AR6$ has higher
priority than $AR10$ and $AR10$ fails, but the framework ap-
plies conservative adaptation, a predefined EvoReqs opera-
tion for $AR10$ could be Replace($AR10_{1day}, AR10_{2days}$).
This way we acquire compensation in accordance with re-
quirements set by the stakeholders.

Given the fact that the changes to parameter values do not
take effect immediately and in the meantime more failures
may take place, it is important to apply a form of synchro-
nization to the adaptation process. In Figure 2 the Avoid
Synchronization Conflicts and the AR1 satisfy this need.
This goal states that when a failing indicator is being fixed
no parameter can be tuned in a way that will affect the fail-
ing indicator negatively. For example, if $AR8$ is failing and
the adaptation framework increases $RfM$ to fix it. How-
ever, this change may require several time to take effect
and before that happens, $AR2$ fails. In order to fix $AR2$ the
parameter $VP2$ must be decreased, but that would affect
negatively $AR8$ before the latter has been fixed. To avoid
such situations that could lead the adaptation process into
non-converging adaptations, we set as a requirement not to
apply changes that affect unresolved indicators for the shake
of fixing other failures.

4. ADAPTATION PROCESS FOR MULTIPLE
FAILURES

In the previous section we presented a set of features
such as prioritization and compensation mechanisms that
can help us handle multiple concurrent failing indicators
and compose dynamically adaptation strategies. This sec-
tion describes the additions to the Zanshin framework that
implement these features and explains the steps that the
adaptation process carries out.

Figure 4 depicts the conceptual architecture of the ex-
tended Zanshin framework. A Monitor Component exam-
ines the log files that are produced at runtime from the base
system if any failing $AwReqs$ are detected the Failure Man-
ger and the Adaptation Manager are informed. The Failure
Manager groups failing indicators according to the presence
of conflicts with other indicators and informs the Decision-
Maker component. The Adaptation Manager is responsible
for configuring the adaptation process by monitoring the re-
quirements model such as the one in Figure 2. When an
$AwReq$ fails, it selects an adaptation strategy for this fail-
ure. Then, it informs the Decision-Maker about the selected
parameter values of the adaptation process goal model in
order to perform the trade-offs accordingly. The Decision-
Maker exploits input from other components and the indica-
tors’ value derived from the AHP to select which indicators
should be tuned and which should be compensated. The
Strategy Manager converts decisions to adaptation strat-
gies by putting together all the required operations. Finally,
the Adapt component executes the operations that are pre-
scribed in the adaptation strategy.

To give a better understanding of how the framework op-
erates, the diagram of the Figure 3 presents every step of
the adaptation process. The steps are:

1. All the $AwReqs$ failures are collected by the failure man-
ager;
2. The indicators of the failing $AwReqs$ are separated

with criteria related to the conflicts they may be part of;
3. The decision-maker exploiting the differential relations
and the values assigned to each indicator resolves any
conflicts that may exist by deciding what action should
be performed;
4. For every indicator that is selected to be fixed there
might be more than one parameter available and one
parameter may be able to fix more than one selected in-
dicator. The framework selects the minimum number
of parameters that should be tuned to fix the chosen
indicators.
5. The values for the selected parameters are calculated;
6. The values of the selected parameters are changed and the
EvoReqs operations for the indicators that can’t be
handled are executed;
7. The framework waits for the changes to take effect;
8. After the wait time the indicators are evaluated again;
9. In each cycle, the process learns from the outcome of
this change and the recorded data could be used to
derive quantitative relations among indicators and pa-
rameters;
10. Finally, if there are no more failing $AwReqs$ after the
evaluation the process terminates successfully.
11. Otherwise, the framework will look for violations on
the requirements for the adaptation process and if any
are found the defined adaptation strategy will be exe-
cuted.

The strategy manager composes a new strategy with all
the actions that will apply the new values to the param-
ters. These actions are executed by Requirement evolution
operations on the target system. The framework waits for
an amount of time for the changes to take place and then
evaluates the indicators that were treated. There is a step
that the framework is performing learning in order to derive
quantitative relations among parameters and indicators, but
it’s part of our future research agenda and is not examined in
this paper. Finally, the algorithm terminates if all the fail-
ing $AwReqs$ are fixed and if not the policy manager controls
Adaptation Manager is listed in Table 5. The Evolution API is the one that groups failing indicators. The Policy if there are any failures. The status of the adaptation requirements and switches conflict(s). For our case study the assigned EvoReq fixed by changing a parameter value due to the presence of sites for our framework’s input is to assign to each AwReq the result in Table 4. The final step for having all the prereqs don’t conflict with each other and the sum of their priority values is the maximum. This is a well-studied NP-hard combinatorial optimization problem [11] for which greedy algorithms have been proposed [15] and perform sufficiently well. We note that as a task of our research agenda is to improve the decision-making performance of the framework by exploiting log data and deriving quantitative relations among indicators and parameters through. Then, faster quantitative optimization approaches will be applicable.

5. EVALUATION

This section explains how the Qualia+ mechanism works through the Meeting Scheduler case study we introduced in Section 2. Then we demonstrate several cases that the older version of Zanshin wouldn’t be able to handle as effectively as the extended one does.

5.1 Meeting Scheduler Case Study

The first step for building an adaptive system that will be managed by our proposed framework is to perform system identification and elicit the differential relations among AwReqs and the parameters of the system’s goal model. From the goal model of the Meeting Scheduler depicted in Figure 1 we elicit a set of differential relations presented in Table 3. We then apply AHP as discussed earlier and show the result in Table 4. The final step for having all the prerequisites for our frameworks input is to assign to each AwReq an EvoReq operation to be executed in case it cannot be fixed by changing a parameter value due to the presence of conflict(s). For our case study the assigned EvoReq operations are listed in Table 5.

Figure 4: Zanshin Architecture

| order (RF) : listonly < short < full | (2) |
| order (VP2, AR10) : partner < hotel < local | (3) |
| \[ \Delta (I_1/RF) < 0 \] | (4) |
| \[ \Delta (I_2/RfM) < 0 \] | (5) |
| \[ \Delta (I_3/VP2) < 0 \] | (6) |
| \[ \Delta (I_4/FhM) < 0 \] | (7) |
| \[ \Delta (I_5/RfM) > 0 \] | (8) |
| \[ \Delta (I_6/VP2) > 0 \] | (9) |
| \[ \Delta (I_7/MCA) > 0 \] | (10) |
| \[ \Delta (I_8/VP3) < 0 \] | (11) |
| \[ \Delta (I_9/RF) > 0 \] | (12) |
| \[ \Delta (I_{10}/Hm) > 0 \] | (13) |
| \[ \Delta (I_6/VPA) \{false \rightarrow true\} > 0 \] | (14) |
| \[ \Delta (I_6/MCA) < 0 \] | (15) |
| \[ \Delta (I_6/VP1) < 0 \] | (16) |
| \[ \Delta (I_6/VP3) < 0 \] | (17) |
| \[ \Delta (I_7/VPA) \{false \rightarrow true\} < 0 \] | (18) |
| \[ \Delta (I_8/RfM) \{0, enough\} > 0 \] | (19) |
| \[ \Delta (I_8/VP2) > 0 \] | (20) |
| \[ \Delta (I_9/MCA) > 0 \] | (21) |
| \[ \Delta (I_9/VP3) > 0 \] | (22) |
| \[ \Delta (I_{10}/RF) < 0 \] | (23) |
| \[ \Delta (I_{10}/FhM) < 0 \] | (24) |
| \[ \Delta (I_{10}/VP1) > 0 \] | (25) |
| \[ \Delta (I_{10}/VP2) > 0 \] | (26) |
| \[ \Delta (I_{10}/VP3) > 0 \] | (27) |

Table 3: Differential relations elicited for the Meeting Scheduler example [18]
In our previous work [18] we state that some EvoReq operations can act either at instance level or class level. For example, when a requirement R is replaced by a requirement R’ at an instance level, it means that future runs of the base system will use R, not R’. On the other hand, a class-level change means that subsequent executions of the base system will use R, not R’. As presented in Table 5, AR1, AR2, AR3, AR6, AR8 and AR10 can be replaced by other requirements with weaker quality constraints. For example, for Good participation, instead of expecting 90% participation we could lower expectations to 80%. For the AwReqs AR1, AR4 and AR5 we don’t weaken requirements but rather postpone dealing with them, using the Retry(time) operation. For AR7 and AR9 we use the EvoReq operations Warning() and Abort() respectively. The first one prints a warning message and the second one suspends the requirement altogether.

Now that all the required input for Zanshin has been specified we present a case of multiple failures and how these are resolved. The adaptation requirements for the framework are those presented in Figure 2 and the predefined value of VP1 is \textit{Adapt Optimistically}. The monitor component checks periodically every 1 hour if there are any failures. In the first scenario the monitor detects failures of AR1, AR2. Then the Failure Manager collects the parameters than can tune failing indicators. According to Table 3 for AR1 the only option is to decrease RF (required fields to organize a meeting) and for AR2 either decrease RFM (Rooms for Meetings available) or decrease VP2. There are though priority conflicts with non-failing AwReqs a) AR1 conflicts with AR6 and b) AR2 conflicts with AR4 and AR8. The Decision-Maker takes into account the adaptation goal \textit{Adapt Optimistically} and ignores the priority conflicts. Therefore, the Strategy Executor composes an adaptation strategy that executes two operations \textit{decrease(RF)} and \textit{decrease(RFM)}. The framework will wait for the effects to take place and then examines if the indicators still need improvement. The previous AwReqs are not failing anymore, but AR4 and AR8 are adversely affected as they are now failing. Moreover, the Adaptation Manager because of these new failures switches to \textit{Adapt with Compensation}. The Decision Manager then decides to increase the parameter RFM to improve AR4 and AR8 and executes the assigned EvoReq operation for AR2 since it has the lowest priority and no other parameter to be reconfigured. The result is a new strategy with the operations \textit{increase(RFM)} and \textit{Replace(Ar2, Ar2_200euro)}. The outcome is that the new AR2 is not failing anymore and AR4 and AR8 are not failing.

### Table 4: Priority Values of AwReqs

<table>
<thead>
<tr>
<th>AwReq</th>
<th>Priority Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR6</td>
<td>1.63</td>
</tr>
<tr>
<td>AR5</td>
<td>1.58</td>
</tr>
<tr>
<td>AR4</td>
<td>1.17</td>
</tr>
<tr>
<td>AR1</td>
<td>1.09</td>
</tr>
<tr>
<td>AR8</td>
<td>0.93</td>
</tr>
<tr>
<td>AR2</td>
<td>0.8</td>
</tr>
<tr>
<td>AR3</td>
<td>0.7</td>
</tr>
<tr>
<td>AR7</td>
<td>0.76</td>
</tr>
<tr>
<td>AR9</td>
<td>0.64</td>
</tr>
<tr>
<td>AR10</td>
<td>0.6</td>
</tr>
</tbody>
</table>

### Table 5: EvoReq operations for AwReqs

<table>
<thead>
<tr>
<th>AwReq</th>
<th>EvoReq operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR1</td>
<td>Retry(50000ms)</td>
</tr>
<tr>
<td>AR2</td>
<td>Replace(Ar2, Ar2_200euro)</td>
</tr>
<tr>
<td>AR3</td>
<td>Replace(Ar3, Ar3_14d)</td>
</tr>
<tr>
<td>AR4</td>
<td>Retry(1day)</td>
</tr>
<tr>
<td>AR5</td>
<td>Retry(10000)</td>
</tr>
<tr>
<td>AR6</td>
<td>Replace(Ar6, Ar6_80%pirt)</td>
</tr>
<tr>
<td>AR7</td>
<td>Warning()</td>
</tr>
<tr>
<td>AR8</td>
<td>Replace(Ar8, Ar8_14d)</td>
</tr>
<tr>
<td>AR9</td>
<td>Abort()</td>
</tr>
<tr>
<td>AR10</td>
<td>Replace(Ar10, Ar10_3days)</td>
</tr>
</tbody>
</table>

### 5.2 Improved Adaptation

The previous version of Zanshin and Qualia adaptation mechanism were ignoring the fact that there are cases where multiple failures cannot be handled independently by treating individually and sequentially indicators of failing AwReqs. To test our approach we implemented a simulation of the meeting scheduler example. The participants were implemented as agents with their own calendars and the rooms for the meetings registered to our implementation’s database have the same characteristics with the meeting rooms of the University of Trento and its collaborating institutions. To test our system we initialize the calendars of every participant in order to create more realistic conditions. Then the user of the system has the role of the meeting organizer and initiates meeting requests specifying the room requirements and the participants. The participants may have conflicts with their own schedules or not. A monitoring mechanism is measuring every indicator after every meeting and updates it’s status as failing or healthy. For comparison reasons we ran the same simulations using Qualia and Qualia+ and below we demonstrate some of the failing scenario that took place and how each algorithm dealt with them.

**Scenario 1:** The monitor detects failures for AR4 and AR8.

**Qualia:** The framework will treat first the failure that was detected first, in this case AR4. The parameter RFM is increased due to the differential relation (7) of Table 3. If after the change AR4 is not failing a new adaptation session starts for AR8 increasing VP2 due to the differential relation (18). When AR8 ceases to fail as a consequence of the parameters increment AR2 fails because of the negative impact the changes (differential relations (4) and (5)). Then Qualia would decrease again either RFM or VP2 (or both) in order for AR2 to recover. It is obvious that such an adaptation mechanism will fall into an infinite loop doing and undoing the same changes.

**Qualia+:** The framework is instructed how to adapt in these cases using AdReqs as described in the example of the previous subsection.

**Scenario 2:** The monitor detects failures for AR5, AR6 and AR9.
Qualia: The framework treats again the failures sequentially changing parameters that would result again in an infinite loop.

Qualia+: The Failure Manager finds MCA (maximum conflicts allowed among the participants’ time-schedules) as only available parameter which means that the Decision Maker has to choose between increasing MCA to improve AR5 and AR9 and worsen AR6 or decrease MCA to improve AR6 and worsen AR5 and AR9. The choice is based on the priority values of the indicators shown in Table 4 and since (1.58±0.64>1.63) AR5 and AR9 are preferred. The finally result would be a strategy with the operations: a) increase(MCA) and b) Replace(Ar6, Ar6,80%pr1). This way we compensate for not improving AvReq while meeting adaptation requirements.

We note that adaptation strategies could have also been composed defining rules that specify in what order should EvoReqs operations be executed and under what circumstances. However, such rules are outside the scope of requirements and hard for stakeholders to conceptualize, and therefore define. Qualia+ allows stakeholders to define the policies by which such conflicts should be resolved during the adaptation process. Moreover, the dynamic composition of adaptation strategies means that the adaptation process does not need to go off-line when adaptation requirements are changed.

6. RELATED WORK

In the field of software-adaptation a variety of approaches have been proposed to handle multiple objectives that need to be achieved. First, Rainbow[6] exploits Utility Theory to perform trade-offs among conflicting objectives maximizing the overall utility of the system. In the context of the same work, improved adaptation strategies are proposed using the concept of preemption [13]. The revised adaptation mechanism schedules better available strategies and can interrupt them when needed if new failures, more important, come into play. These strategies intended to be based on human experience and expertise of a control-based process. Consequently, there is no guarantee that after the addition or removal of one or more requirements they will still be effective. On the other hand, the basic idea of our work is to automate the composition of the adaptation strategies based on stakeholder requirements and preferences.

The work of Kramer and Magee [9] proposes another Architecture-based solution for self-adaptive systems. Adaptations in this framework are determined at three layers. The bottom layer, called Component Control, monitors the base-system and detects failing components. When a failure is noticed a request for adaptation actions is sent to the upper layer, called Change Management. This layer is responsible for an adaptation plan that will reconfigure and restore the failing system. If a plan is not available then the top layer, called Goal Management is invoked to produce new adaptation plans when new goals are introduced to the system. In this work, as in ours, adaptation strategies are composed dynamically. Our proposal attempts to go farther by exploiting requirements models that prescribe not only how the base-system should operate but also how it should adapt, even if there are no available adaptation actions, through EvoReqs. On the other hand architecture models facilitate software adaptation because they offer adaptation options that are simply not there in the requirements models we use. In our previous work [1] we advocate the combination of Requirements-based and Architecture-based approaches to better support adaptation. This is also part of our research agenda.

Another Requirements-based approach is RELAX [23], that captures uncertainty declaratively with modal, temporal and ordinal operators applied over SHALL statements (e.g., “the system SHALL ... AS close as possible to ...”). A similar approach, but based on the goal-oriented language KAOS [3], is FLAGS [2]. This approach extends the linear temporal logic (LTL) used in KAOS with fuzzy relational and temporal operators, allowing some goals to be satisfied even if values are close to but not equal to the desired ones. FLAGS also proposes an operationalization of its models in a service-oriented infrastructure. Although FLAGS deals with conflict resolution and synchronization during the adaptation process as our work does, the differential relations, the prioritized requirements and AdReqs offer a more pragmatic basis for stakeholders to express their requirements on the adaptation process.

Multiple failures are also handled by Vromant et al in [22]. This work adopts and extends the Monitor-Analyze-Execute-Plan (MAPE) adaptation model focusing on interacting feedback loops. The baseline exemplar is a traffic monitoring system that is composed of several groups of agent-cameras that depend on each other to operate and provide accurate results. When several agents are not operating properly, several adaptation loops are being executed and co-ordinated in order to produce effective adaptation plans. Even though the approach presents an interesting way to adaptation mechanism to schedule adaptation actions, the focus is only on detecting which agents are not responding and how the the nodes of the system should be regrouped so that the system will keep providing the required service. In other words the dependencies are among system components and not the requirements as in our case.

Another Requirements-based and control theoretic approach is presented in [12]. In this work the authors implement a PID controller that monitors the set values of the satisfaction of goal model’s softgoals and when the output declines finds a different configuration over the goal model. A preference ranking is used to assist the trade-off process among conflicting softgoals. This configuration is a result of reasoning supported by the OpenOMU tool [16]. The implementation of the PID controller enhances the precision to the adaptation process, however the approach takes into account only failing softgoals. Moreover, there is no consideration about how lower priority failures are to be treated, while in our proposal we introduce the concept of compensation to achieve that. Nevertheless, this work is closest to the goal-oriented requirements framework we are using here.

7. CONCLUSIONS

The main contribution of this paper is a requirements-based adaptation mechanism that can handle multiple concurrent requirements failures. To accomplish this, we have extended the Zanshin framework with two basic new features. The first is the concept of AdReqs, which are requirements about the adaptation process itself. Like all requirements, these come from the stakeholders and define policies the adaptation process has to comply with. The second fea-
ture is a decision making mechanism that takes into account AdReqs to decide which requirements should be improved and which have to be compensated temporarily or permanently by EvoReqs operations. The new adaptation mechanism, called Qualia+, makes it possible for stakeholders to reflect their needs and preferences for the adaptation process by assigning priorities and compensation operations to base system requirements. Moreover, the fact that adaptation strategies are composed dynamically allows stakeholders to change AdReqs during system operation.

Our proposal needs to be further evaluated to ensure its effectiveness on several case studies. Moreover, we plan to extend Zanshin so that it can learn during its operations quantitative relations between indicators and control parameters, to replace qualitative relations provided by requirements engineers and domain experts.

8. ACKNOWLEDGMENTS
This work has been supported by the ERC advanced grant 267856 “Lucretius: Foundations for Software Evolution” (April 2011 – March 2016, http://www.lucretius.eu).

9. REFERENCES